A Distributed Data Management Middleware for Data-Driven Application Systems*

Stephen Langella, Shannon Hastings, Scott Oster, Tahsin Kurc, Umit Catalyurek, Joel Saltz

Department of Biomedical Informatics,
The Ohio State University
Columbus, OH, 43210
{langella,hastings,oster,kurc,umit,jsaltz}@bmi.osu.edu

Abstract

A key challenge in supporting data-driven scientific applications is the storage and management of input and output data in a distributed environment. In this paper, we describe a distributed storage middleware, based on a data and metadata management framework, to address this problem. In this middleware system, applications define the structure of their input and output data using XML schemas. The system provides support for 1) registration, versioning, management of schemas, and 2) management of storage, querying, and retrieval of instance data corresponding to the schemas in distributed databases. We carry out an experimental evaluation of the system on a set of PC clusters connected over wide- and local-area networks.

1 Introduction

Advanced data acquisition technologies (e.g., high resolution MR scanners, high throughput microarray analyzers, and advanced satellite sensors) have made significant progress over the last decade. As a result, the amount and types of data to be referenced and managed have rapidly increased. Moreover, distributed computing technologies have enabled applications to carry out simulation of large and complex numerical models on machines in distributed environments, generating large volumes of simulation output to be analyzed. As a result, data-driven scientific applications have become an important application class that can benefit from distributed storage and computing. Examples include applications that carry out parameter sweep studies [9], optimization strategies that evaluate hundreds (even thousands) of scenarios in the course of searching for the optimal solution to a given question [20, 18], and applications that consist of large workflows [3, 12, 10]. Input datasets to these applications can be stored in databases or files. A user-defined predicate specifies the set of data elements (as database rows, files, or parts of files) to be processed. After the data is processed, output can be stored in the environment as new datasets, which can be further processed by other applications.

In order to support a scalable number of applications that generate and reference various structures of different data types and data dispersed across disparate locations, mechanisms for communicating and sharing data are needed. The environment must support the ability for applications and services to store, discover, and retrieve data in a general and flexible way. With such mechanisms, complex applications can be implemented from loosely coupled components, services, or applications. Sharing of datasets and documents among related applications can also be useful for data provenance [12] and for efficient execution of multi-query workloads [28, 2].

In this paper, we develop a data storage and management middleware that is *distributed* – system components run on networked collections of PCs and PC clusters, *searchable* – input and output instance data is stored in databases created from schemas and can be queried by applications, and *shareable* – multiple instances of an application or multiple applications with different input and output schemas can concurrently store and retrieve data in the system. The middleware supports the ability to 1) create schemas which describe the input data models used by the data analysis application, 2) register these data models so that other researchers can reference them in their applications, and instance data based on these models can be queried, 3) manage and version the models as new data types are added or deleted, and 4) integrate, create, and manage databases

*This research was supported in part by the National Science Foundation under Grants #ACI-9619020 (UC Subcontract #10152408), #EIA-0121177, #ACI-0203846, #ACI-0130437, #ANI-0330612, #ACI-9982087, Lawrence Livermore National Laboratory under Grant #B517095 (UC Subcontract #10184497), NIH NIBIB BISTI #P20EB000591, Ohio Board of Regents BRTTC #BRTT02-0003.

that conform to these data models. In our system, the input and output datasets of an application are defined by XML schemas. The system provides a distributed storage and data management structure for applications to query and retrieve input data (which may have been generated by instruments or other applications) and to store their outputs in databases conforming to the schema definitions.

The system is implemented as a services-based middleware framework, called Mobius [15]. The architecture of Mobius is motivated by the activities of the Data Access and Integration Services group at Global Grid Forum [4, 5] and by earlier work [19]. Mobius consists of several services and underlying protocols that support distributed creation, versioning, management of data models defined by XML schemas, on-demand creation of distributed databases, federation of existing databases, and querying of data in a distributed environment. We describe the implementation of the system and present an experimental evaluation under varying workload characteristics on a set of PC clusters connected over local- and wide-area networks.

2 The Mobius Framework

Mobius consists of three core services: Global Model Exchange (GME), Metadata and Data Instance Management (Mako), and Data Translation Service (DTS). Mobius services employ XML schemas to represent metadata definitions or data models and XML documents to represent and exchange metadata instances or data instances. In this paper, we provide a description of the GME and Mako services that are relevant to our target application environment. Further details on other aspects of Mobius can be found in a previous work [15], in which we describe how the system can be used for integration of data from disparate data sources.

2.1 Global Model Exchange

The Global Model Exchange (GME) is responsible for storing and linking data models as defined inside namespaces in the distributed environment. The GME enables other services to publish, retrieve, discover, deprecate, and version metadata definitions. GME services are composed together in a domain name server-like architecture representing a parent-child namespace hierarchy wherein parents act as authorities for children and provide them with a sub-namespace. When a schema is registered in GME, it is stored in under the name and namespace specified by the application and is given a version number. We refer to the tuple consisting of the schema's name, its namespace, and its version number as the global name id (GNI) of the schema.

The GME provides model version and model-to-model dependency management. For instance, if a user service publishes a model to the GME and later the model is modified and republished, the model will automatically be versioned. The GME protocol provides a mechanism for stating the exact version of the model that is requested. A model can also contain types defined by other models or references to types contained in other models, and can be assured that the referenced entities exist. This reference integrity might be considered the largest requirement for a GME that the current use of a URL does not provide. The role of the GME in the greater picture is to ensure distributed model evolution and integrity while providing the ability for storage, retrieval, versioning, and discovery of models of all shape, complexity, and interconnectedness in a distributed environment. A future extension of the GME service architecture would be to support semantic model storage, versioning, and querying. With semantic model support, one could imagine being able to pose the question, "Are there any models published anywhere in the environment that have something to do with reservoir simulation?".

2.2 Mako

Mako is a service that exposes data resources as XML data services through a set of well-defined interfaces based on the Mako protocol. A data resource can be a relational database, an XML database, a file system, or any other data source. Data resource operations are exposed through a set of well-defined interfaces as XML operations. For example, once exposed, a relational database would be queried through Mako using XPath as opposed to querying it directly with SQL. Mako provides a standard way of interacting with data resources, thus making it easy for applications to interact with heterogeneous data resources.

2.2.1 Mako Architecture

Clients interact with Mako over a network; the Mako architecture illustrated in Figure 1 contains a set of listeners, each using an implementation of the Mako communication interface. The Mako communication interface allows clients to communicate with a Mako using any communication protocol. For example, if the communication interface is implemented using Globus [14] security, clients may communicate with Mako using the Globus Security Infrastructure (GSI). Each Mako can be configured with a set of listeners, where each listener communicates using a specified communication interface.

When a listener receives a packet, the packet is materialized and passed to a packet router. The packet router determines the type of packet and decides if it has a registered handler for processing a packet of that type. If such a

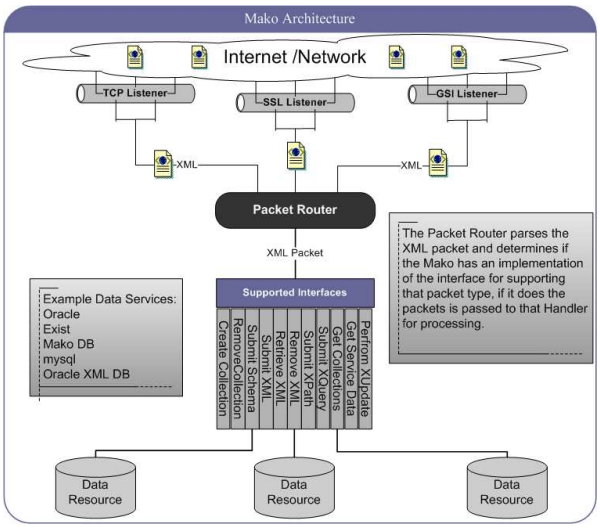


Figure 1. Mako Architecture

handler exists, the packet is passed on to the handler which processes the packet and sends a response to the client.

2.2.2 Mako Protocol

The Mako protocol consists of a set of packet types that enable clients and applications to interact with a Mako. This section will present an overview of the relevant areas of the Mako protocol.

Collection Management. In the context of XML databases, the common nomenclature for referring to a group of related instance documents is a Collection. This is very similar to the relational database concept of a database. The concept diverges slightly, in that collections can have sub collections, and collections may not have a single schema (or any at all) associated with them. The Mako protocol supports three collection management operations, creation of a collection, removal of a collection, and the ability to get a list of sub collections for a given collection.

Schema Management. Each collection in Mako can be restricted to accepting XML documents from a set of certain types or namespaces. This is accomplished by specifying a set of XML schemas which any submitted XML document must be validated against. The Mako protocol provides a method for adding and removing schemas to and from an XML collection as well as a method for obtaining a list of schemas that a collection supports.

Document Management. The Mako protocol defines methods for submitting, updating, removing, and retrieving XML documents. Upon submission, Mako assigns each entity a unique identifier. Documents, or subsets of XML documents, can be retrieved by specifying their unique identifier. Each element in an XML document is given an identifier, making any subset of a document uniquely addressable. The Mako protocol also allows the level of retrieval to be specified. For example, if we think of an XML document as a tree, then given a unique identifier, one would be able to specify how many levels of children should be included in the materialization of the document. Elements containing children that are below the height specified would not be included in the materialization, however references to their immediate children would be included. This feature becomes quite valuable when working with large datasets, in that full documents do not need to be materialized just to view partial results. It also allows one to build a demand driven Document Object Model (DOM) on top of the protocol. In general such a feature improves application performance by allowing the application to specify how data is materialized.

XML documents can be removed by specifying their unique identifier or by specifying an element identifying XPath [6] expression. XML documents that reside in a Mako can be updated using XUpdate[1].

Querying. Mako currently provides query support through the use of XPath [6]. XPath is a declarative language for selecting subsets of XML documents' elements, attributes, and text sections. It is similar to SQL in that it provides a facility for selecting data based on a set of criteria. However, it differs from SQL in that it contains rich facilities for expressing criteria in terms of the inherent hierarchy associated with an XML document.

2.2.3 Mako Handlers

For each packet type in the Mako protocol, there is a corresponding handler to process that packet type. In order to abstract the Mako infrastructure from the underlying data resource, there is an abstract handler for each packet type. Thus, for a given data resource, a handler extending the abstract handler would be created for each supported packet type. This isolates the processing logic for a given packet, allowing a Mako to expose any data resource under the Mako protocol. Our current Mako implementation contains a handler to expose XML databases that support the XMLDB API[1]. It also contains handler implementations to expose MakoDB. MakoDB is an XML database built on top of MySQL [21], and it is optimized for interacting in the Mako framework. There are multiple reasons for building our own XML database; the summary of them is that we want to have total control of the data storage. This allows us an efficient way to uniquely identify pieces of XML documents. That is, any node within an XML document can be addressed as a separate XML document. Having this capability enables one to build XML documents containing sub-pieces of existing XML documents. We refer to this

[1] http://www.xmldb.org

concept as virtual inclusion, which is discussed in the next section. Technical details of MakoDB are outside the scope of this paper and will be discussed in a forthcoming paper.

Data services can easily be exposed through the Mako protocol by creating an implementation of the abstract handlers. Since there is an abstract handler for each packet type in the protocol, data services can expose all or a subset of the Mako protocol. Once handler implementations exist, Mako can be easily configured to use them. This is done in the Mako configuration file, which contains a section for associating each packet type with an appropriate handler.

2.2.4 Global Addressing and Virtual Inclusion

We mentioned earlier that Mako provides a method of uniquely identifying elements contained in XML documents. In reality these elements are uniquely identified across the collection that they reside in. Mako also provides a method of globally addressing XML elements. This is done using the three tuple id, (Mako URI, collection, elementId). Being able to globally address entities within Mako provides several advantages; such as allowing data to be federated across multiple Makos via *virtual inclusion.*

Virtual inclusion is a reference within an XML document to another XML document. This means that Mako allows XML documents to be created that may contain references to existing XML documents or elements, both local and remote. In this way, an XML document can be distributed and stored across multiple Mako servers by storing subsections of the document remotely and integrating them with references. This ability is critical in enabling large data documents to be partitioned across a cluster while still maintaining the single document semantics of a model.

2.3 Virtual Mako

Utilizing Mako's component architecture, alternate protocol handlers can be installed in a Mako server to enable it to utilize remote Mako instances. The Virtual Mako is a collection of protocol handler implementations that extend the operation of the data services to operate on an administrator defined set of remote Makos. It maps a number of Virtual Collections to a set of remote collections. This simplifies the client-side complexity of interfacing with multiple Makos by presenting a single virtualized interface to a collection of federated Makos.

3 Application Implementation and Execution

In our system, the structures of an application's input and output data are specified by XML schemas. The schema can be the same as a schema already registered in the system, or it can be created by versioning an already existing schema by applying modifications to it. The schema can also be a composition of new elements and references to multiple existing schemas. The schemas are managed by Mobius GME. For instance, if an application takes images as input, an image data schema may define metadata associated with an image, such as the type of the image, study id for which this image was acquired, date and time of image acquisition, etc. An instance of the schema corresponds to an image dataset with images and associated image metadata stored across multiple storage systems running data management servers (i.e., Makos).

The requirement on application developers to define schemas for their data types may at first seem like too much of a burden. However, the XML schema definition language provides a large number of primitive data types, which makes the task of defining schemas for simple datasets trivial. Furthermore, these simple types can easily be composed to form common data structures which can be leveraged across multiple application schemas. It should also be noted that many tools exist which can transform a sample XML document into a corresponding XML schema which can then be generalized to account for variances in the expected datasets.

An application using our system reads input datasets conforming to some schema and generates output datasets, again conforming to a schema. For each input dataset, one or more selection criteria can be specified. The selection criteria defines the subset of data to be processed from input datasets and should be executed as a query into the corresponding input datasets. In this paper, we do not address the issues associated with allocation of resources, instantiation of application instances on those resources, and management of the execution of the application. A number of middleware tools and services exist for this purpose [7, 8, 10, 14, 31] and can be used in conjunction with our data management system. The application can be a parallel or sequential program, a component in a data processing workflow, or a Grid or Web service. We assume that the application runs on machines in a distributed environment and interacts with schema management and data management servers using Mobius protocols. In a cluster environment, multiple Mako servers can be instantiated on each node, and one or more GMEs can be instantiated to manage schemas. In some cases, multiple instances of an application can be concurrently executed in the environment. For example, if processing of an input element/file can be done independent of processing of other elements, multiple instances of the same application can be executed in order to achieve high data processing throughput.

When the application (or an instance of the application) outputs a data instance, it can send the instance to any Mako server to which the application is authorized to write. The

output data instance identifies the GNI of the schema to which the data adheres. When a Mako server receives a data instance, it firsts checks if the schema is locally stored and a database has already been created from this schema. If the schema does not exist locally the Mako contacts a Mobius GME to retrieve the corresponding schema definition associated with the particular GNI. After the schema definition is retrieved, it is cached locally and a database conforming to the schema definition is created. Note that a Mako server can control multiple databases distributed across multiple machines. Hence, a schema can be instantiated on a distributed database. Once the database has been created, data instances can be inserted into, retrieved from, updated, and queried from the database.

An application programming interface allows for an application to partition a large byte array into smaller pieces (chunks), associate meta-data description with each piece (represented by a schema), and distribute these pieces across multiple, remote Mako servers. For example, if the byte array corresponds to a large 2D image, each chunk can be defined as a rectangular subsection of the image. The meta-data associated with a chunk can be the bounding box of the chunk, its location within the image, the image modality, etc. The data can be distributed across an application-specified set of Mako servers or by a Virtual Mako server, which distributes the data using a data declustering algorithm[2] across multiple Mako servers. The API also enables the application to build an index document that contains the references to the chunks, and submit it to a designated federation of Makos, using the Virtual Mako, responsible for storing data index documents. When the application wants to retrieve the byte array, it can query for the index document or query chunks using the schemas associated with the chunks. When the index document is retrieved, the API allows the application to retrieve individual chunks one by one, in groups, or completely reconstruct the entire byte array.

4 Related Work

Several types of middleware systems have been developed for implementing and executing applications in distributed environments [10, 7, 26, 9, 12, 8, 14, 24, 13, 31]. The *Everyware* toolkit developed by Wolski et al. [30] enables applications to transparently use computational resources in the Grid. While Everyware targets compute-intensive applications, our middleware is focused on storage and management of metadata and data instances for persistent data caching and sharing. The Storage Resource Broker (SRB) [26] provides unified file system-like interfaces to distributed and heterogeneous storage systems. Unlike SRB, we allow on-demand creation of databases and data stored in Mobius can be searched using query predicates. The Distributed Parallel Storage Server (DPSS) [16, 29] project developed tools to use distributed storage servers to supply data streams to multi-user applications in an Internet environment. Unlike DPSS, which is designed as a block-server, data in Mobius is managed as semi-structured documents conforming to schemas. This allows for more complex data querying capabilities, schema validation, and database creation from complex schemas.

Andrade et al. [2] develops a distributed semantic caching system that allows proxies interspersed between application clients and application servers in order to speed up execution of related queries by exploiting cached aggregates. It also enables an application to explore the parallel capabilities of application servers. ActiveProxy-G requires integration of user-defined operators in the system to enable caching of application-specific intermediate results and searching of cached results. Our middleware is designed to work on data that can be described as a semi-structured XML document with a schema.

The Chimera system [12] implements support for establishing virtual catalogs that can be used to describe how a data product in an application has been derived from other data. The system allows storing and management of information about the data transformations that have generated the data product. While Chimera is designed for management of provenance of data generation, we focus on storage and management of metadata and metadata instances.

Our system has some similarities to ”tuple spaces” [1] in that data elements stored in the system could be treated as highly descriptive tuples, and addressed by their content and type. Additionally, the system supports distributed and shared storage for data elements. Our data types and querying capabilities, on the other hand, are extremely expressive as we utilize the rich structural descriptions of XML Schemas and query using advanced XML query languages.

The IBP [22] framework is a common framework for storing and retrieving large amounts of data. IBP does have some similarities to our infrastructure from the storage and retrieval of possibly large data quantities with a uniform protocol. IBP could be used as the storage and delivery protocol for large data objects which travel through our framework. Distributed data storage, discovery, and retrieval are also supported by distributed file systems and peer-to-peer content delivery networks [17, 25, 27, 23]. Our work differs in that the middleware system enables applications to define and publish complex data models in order to efficiently store, query, reference, and create virtualized XML views into distributed and interconnected databases.

[2] Currently, round-robin, weighted round-robin (i.e., distribution of data chunks to storage nodes based on their disk bandwidth), and random distribution schemes are supported as default strategies.

5 Experimental Evaluation

We have carried out an experimental evaluation of our middleware framework using 3 PC clusters.

The first cluster, *OSUMED*, is a 24 node cluster that contains single processor nodes with 933MHz PIII processor, 512MB RAM, and single network port. The nodes are connected to each other over a FastEthernet Switch. The second one, *DC*, is a 5-node cluster switched with gigabit ethernet and each node is a dual Xeon 2.4 Ghz processors, 2 GB ram, and 360 GB disk. The third cluster, *MOB*, is an 8-node cluster switched with gigabit ethernet. Each node of this cluster has dual 64bit AMD Opteron processors, 8GB RAM, and 1.5TB disk. The MOB and DC clusters are connected to each other over a Gigabit network. The connection between the OSUMED cluster and the MOB and DC clusters is a shared 100Mbit/sec network. In the experiments detailed in this paper, one Mako server was executed on each node of a cluster. One Global Model Exchange server was instantiated in the environment.

5.1 Model Driven Database Creation

Database creation is a common interaction in an ad hoc distributed data management system. The following set of experiments benchmark model driven database creation using various sizes and shapes of models. The goal is to investigate the cost at the single server to create a database on the fly based on a user defined model. All of the experiments capture the total time to create the database from the model and break down the cost into four categories: time taken for the model to be read from the socket (Socket Read Time), time taken to turn that model in the buffer into a schema object model (Schema Object Model Time), time taken to break down and insert the schema entities into the database (Schema Insertion Time), time taken to extract the schema and use it to build the table space for instance data storage (Table Building Time). The summation of these categories represents the total processing time. These trials where performed on the 5-node Xeon cluster.

Figure 2 shows the cost of model driven database creation using models which vary in height, width, and size (i.e., the number of children of an element, the number of siblings of an element, and the number of elements in the schema). The models vary from 1 to 1000 elements stepping by an order of 10 each trial in either direction of height, width, or height and width. The height models are designed using an XML schema where the elements are either *nested*, one inside the other, creating a *deep* model in the case of height varying models, or *siblings*, one beside the other, creating a *wide* model, in the case of width varying experiments. The graphs show that the model driven database creation time grows respectively with the height

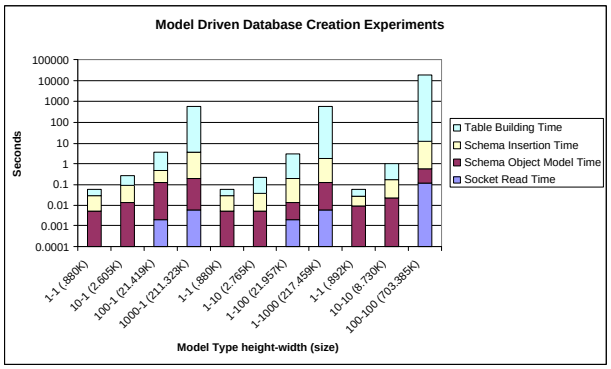


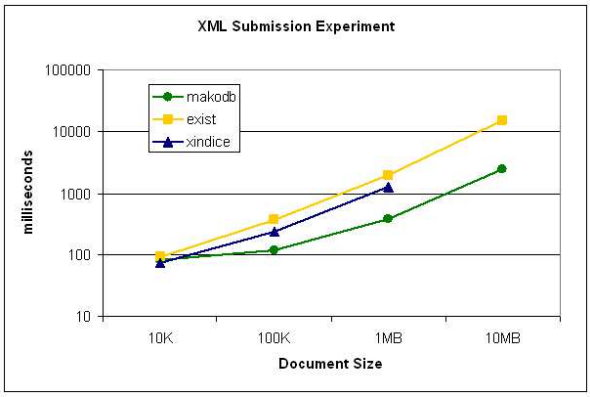
Figure 2. The cost of on-demand database creation from schemas provided by client applications.

or width of the model equally. The graph also shows that the system is bound by the current speed of the table building code and schema model extraction code. Both of these areas are currently unoptimized because they are not large bottleneck areas when looking at the overall use of the system. The data models that we see in most of our application areas (e.g., image analysis applications) are on the order of tens of complex entities per schema. We have also observed that most applications do more reading and writing of data compared with database creation. Nevertheless, we recognize this as a performance bottleneck when using extremely large data models and plan to optimize this in a future work in order to support programmatically maintained data models which may grow to enormous bounds.

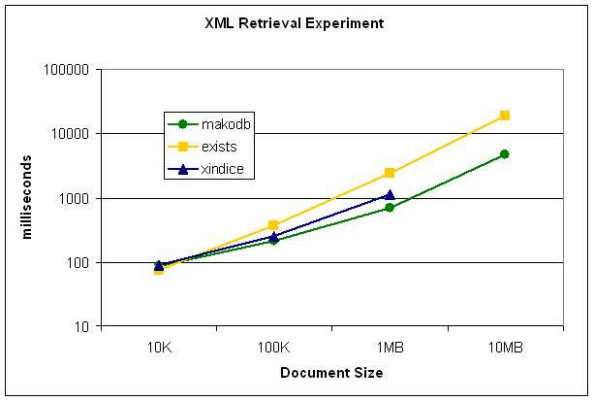
This set of model driven database creation tests conclude that the Mobius Mako database creation cost is independent of the shape of the schema, but is sensitive to its size. Hence, a user can build a model of the data without worrying about fitting the model to a certain shape and without tuning the storage system in which they place the data, to the shape of the model.

5.2 Data Submission, Retrieval and Query Experiments

In our system, data is stored as XML documents in a Mako server. In this set of experiments, we compare the performance of MakoDB with two open source, widely used XML databases, eXist [11] and Xindice [32]. The experiments for the three databases were performed within the Mako environment, thus using the MakoDB handlers for MakoDB and by using the XMLDB handler implementation for Exist and Xindice. The experiments are designed to show the cost of instance data submission, querying, and retrieval as XML documents, where the XML document instances are varied in size and shape according to the data



(a)



(b)

Figure 3. Execution time for (a) data instance submission and (b) data instance retrieval, when the document size is varied.

model to which they adhere. For the submission and retrieval experiments we used two data models. The first model, *Model A*, contains a root element with ten children, each with a child containing text. The second model, *Model B*, is designed to show the cost of both depth and width. It contains 10 elements, each with a single child, that in turn contains a single child. This inheritance is repeated for 10 generations. For each model we generated instance documents of varying sizes (i.e., the size of the text in the document) starting with 10K and going up to 10MB, increasing size by a factor of ten. These trials where performed on the 5-node Xeon cluster. Due to space limitations, we display results for *Model B*; we have observed similar trends with *Model A*.

Figure 3 shows the results for instance submission and retrieval. The results show that for instance submission MakoDB achieves better performance than both Xindice and eXist, especially as the size of the instance documents grow. For MakoDB, most of this increase is because of the increasing amount of data being stored in MySQL. We also noticed an increase in execution time as the number of elements in the document is increased. This can be attributed to several factors. First, since the documents are being validated against a schema, the more elements a document contains the longer its validation takes. Second, each element is modeled separately in the database, which increases the number of database inserts with increasing number of elements. This cost is minimal in our experiments, is proportional only to the number of elements, and does not increase as the amount of data per element increases.

Similar to the submission results, MakoDB showed better performance than both eXist and Xindice in instance retrieval (i.e., retrieval of the entire document) experiments. For MakoDB, retrieval time increases as the number of elements is increased. Again, this can be attributed to the increased number of tables that must be queried. For MakoDB when we compared submission to retrieval, the time to retrieve is higher than that of submission. This is because of the database query time that is required to rebuild all of the relationships between elements in the requested document, from their normalized form back to a materialized document.

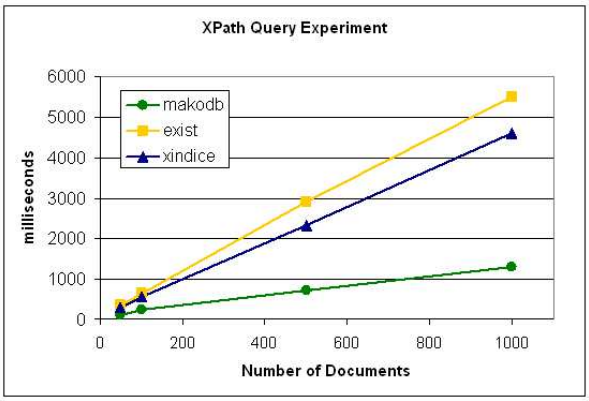


Figure 4. XPath Query Experiment.

To show the cost of performing an XPath query against the three different database implementations, we used a data model designed for representing medical images. The data model contains demographical information and other metadata about the patient from which the image was taken, as well as base64 encoded image data[3] and a base64 encoded

[3]The Mobius protocol supports binary data, which would be used in real applications as it eliminates the processing time needed for encoding and decoding, as well as the 4x increase in data transmission, but for these experiments we used only standard XML as binary objects are not supported in most XML databases.

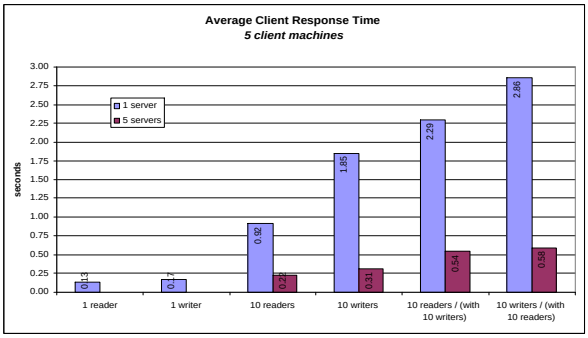


Figure 5. Average response time as the number of applications that read and write data is varied.

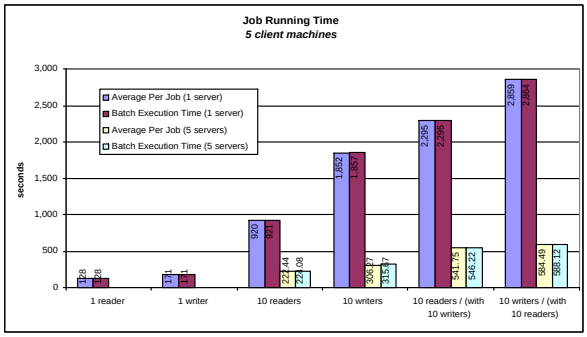


Figure 6. Average job execution time and batch execution time with varying number of client programs.

thumbnail for each image. As an experiment we performed an XPath query that returned all the thumbnail images in the database. The thumbnails are located three levels deep in the XML documents. For this experiment we varied the number of documents in the database. As is seen in Figure 4, MakoDB achieves the best performance of the three databases. MakoDB models each element in an XML document as individual XML documents, thus making it easy to locate and return subsets of XML documents.

5.3 Multi-client/multi-application Scaling Experiments

This set of experiments is designed to evaluate the ability of the system to handle a multi-client environment with different numbers of readers and writers spread across the 5-node Xeon cluster. In each experiment, a client sequentially issues 1000 requests to either read or insert an XML instance document of approximately 10 kilobytes, with 100 elements (adhering to a schema with one root element and 100 child elements). We present a variety of client configurations, showing the single machine, single process base cases, and extending to a fully distributed collection of both readers and writers. When multiple servers are instantiated, each client writes documents to and requests documents from the servers in round robin fashion. The respective times of reading and writing are shown as the number of clients, communicating with a single server and with multiple servers, is varied. It should also be noted that in this set of trials, the nodes in the cluster are used as both servers and clients, but clients never utilized local host servers.

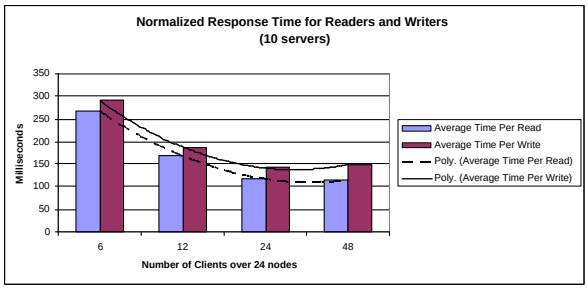
Figure 5 presents the average response time per request, observed by the client. A result which may not be self-evident is the fact that response time on a single server for 10 readers (0.92 seconds) is less than 10 times the 1 reader case (1.3 seconds). This is due largely to the fact that in the single reader case, since the requests are submitted serially, by a single client thread, the server can only process a single request at a time. In the distributed, 10 reader case, the 10 clients (running on 5 nodes) can all submit their requests in parallel, thus allowing the multi-threaded server to process requests concurrently, and thus utilize both CPUs on a server node.

The graph also shows that the response time for individual requests is greatly improved by adding more servers. That is, by using 5 servers instead of 1, we see an improvement of nearly 5 times. This result is echoed in Figure 6, where the average per client completion time is shown, along with the total time to complete all clients. An additional aspect shown in the figure is the time to complete all clients is very close to the average time per client. This shows that the servers are able to service client requests in a fair manner, and all similar clients complete at nearly the same time.

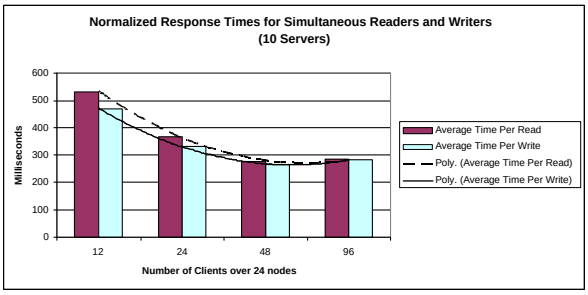
5.4 Multi-Cluster Submission and Retrieval Experiments

This set of experiments was intended to evaluate the Mobius protocol, the Mako protocol, and MakoDB in a multi-user, multi-cluster environment. The experimental setup places servers and varying numbers of clients distributed in a balanced fashion across the three clusters. The clients are each assigned a server to make 1000 serial requests each of size 10KB either reading or writing. The client and server may or may not exist on the same cluster as the clients are randomly assigned to nodes and servers across the three clusters in order to simulate a dynamic multi-user environment. The trials are ran in three different scenarios: clients only reading, clients only writing, and clients half of which are reading and the other half writing. In these experiments, there were 10 servers, each of which is assigned to varying numbers of clients.

Figure 7(a) shows the response times observed by clients



(a)



(b)

Figure 7. (a) Results for instance submission and retrieval with varying number of clients. All the clients write to the servers concurrently and then request documents. (b) Results when half of the clients submit and concurrently the other half retrieve documents with varying number of clients. Both graphs show 2nd degree polynomial fit lines.

reading and writing to the servers. The reading tests were ran independently from the writing tests. That is, all the clients first wrote their documents to the servers. Then, in the second stage, all the clients requested documents from the servers. The time shown is the normalized average response time per request (unit work) as the number of clients is doubled each trial – that is, the normalized response time is equal to the maximum execution time across clients divided by the total number of requests. The graph shows the ability to complete a request in a multi-user environment, as we increase the number of clients. The average response time decreases until the servers are saturated and then begins to rise. The graph shows the optimal location occurs around 24 clients[4].

Figure 7(b) shows the response times when clients read and write simultaneously to the servers. In these trials the readers and writers are ran in parallel and are competing for resources on the clusters. Similar to Figure 7(a), the time shown is the normalized average response time per request (unit work) as the number of clients is doubled each trial. As is seen from the figure, the average response time decreases until an optimal point, then begins to rise. The graph shows the optimal client load for this configuration is around 48 clients.

In these trials, we observed that in the cases where reading and writing trials are run independent of one another, on average the writing clients incur longer response times and do not benefit from parallelism as much as reading clients. When reading and writing are executed concurrently, initially reading clients observe longer response times than writing clients. However, as more clients are added, eventually the writing clients start to observe longer response times. These effects were even more evident in the 5 server trials, as writing client response times overtook reader client times at around 48 clients. We attribute this behavior to locking overheads in MySQL when concurrent updates to the database are executed and to the resource contention of multiple threads on the server which are processing incoming requests. We plan to carry out more detailed examination of potential reasons for the observed system behavior and associated overheads.

[4]We also experimented with 5 servers. The optimal location was 12 clients in that case. A configuration with fewer servers is expected to saturate faster than one with more servers, since each server has to interact with more clients.

6 Conclusions

We have presented a middleware system that provides support for management of metadata definitions (defined as XML schemas) and efficient storage and retrieval of data instances in a distributed environment. The system provides *distributed*, *searchable*, and *shareable* persistent storage for applications. Our initial experimental evaluation shows that the middleware system scales well when the complexity of schemas, data size and the number of clients (up to 96 clients have been tested) are increased. Under multi-client workloads, the system can serve client requests in a fair manner. As a result, all similar clients complete at nearly the same time. Our preliminary results show that the proposed system can provide an efficient mechanism for data-driven applications to cache, share, and asynchronously communicate data in a distributed environment.

References

[1] S. Ahuja, N. Carriero, and D. Gelernter. Linda and friends. *Computer*, 19(8):26–34, 1986.

[2] H. Andrade, T. Kurc, A. Sussman, and J. Saltz. Active Proxy-G: Optimizing the query execution process in the Grid. In *Proceedings of the 2002 ACM/IEEE SC Conference (SC 2002)*. ACM Press, Nov. 2002.

[3] J. Annis, Y. Zhao, J. Voeckler, M. Wilde, S. Kent, and I. Foster. Applying chimera virtual data concepts to cluster finding in the sloan sky survey. In *Proceedings of Supercomputing 2002 (SC2002)*, 2002.

[4] M. Antonioletti, M. Atkinson, S. Malaika, S. Laws, N. Paton, D. Pearson, and G. Riccardi. Grid Data Service Specification. Technical report, Global Grid Forum, 2003.

[5] M. Antonioletti, A. Krause, S. Hastings, S. Langella, S. Malaika, J. Magowan, S. Laws, and N. Paton. Grid Data Service Specification: The XML Realisation. Technical report, Global Grid Forum, 2003.

[6] A. Berglund, S. B. X. WG), D. Chamberlin, M. F. Ferndez, M. Kay, J. Robie, and J. Simon. *XML Path Language (XPath)*. World Wide Web Consortium (W3C), 1st edition, August 2003.

[7] M. D. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, and J. Saltz. Distributed processing of very large datasets with DataCutter. *Parallel Computing*, 27(11):1457–1478, Oct. 2001.

[8] H. Casanova and J. Dongarra. NetSolve: a network enabled server for solving computationa l science problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, 1997.

[9] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS parameter sweep template: User-level middleware for the grid. In *Proceedings of Supercomputing 2000 (SC2000)*, 2000.

[10] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping abstract complex workfbws onto grid environments. *Journal of Grid Computing*, 1(1), 2003.

[11] eXist. *http://exist.sourceforge.net/*.

[12] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *Proceedings of the 14th Conference on Scientific and Statistical Database Management*, 2002.

[13] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*. IEEE Press, Aug 2001.

[14] The Globus Project. http://www.globus.org.

[15] S. Hastings, S. Langella, S. Oster, and J. Saltz. Distributed data management and integration: The mobius project. In *GGF Semantic Grid Workshop 2004*, pages 20–38. GGF, June 2004.

[16] W. Johnston, J. Guojun, G. Hoo, C. Larsen, J. Lee, B. Tierney, and M. Thompson. Distributed environments for large data-objects: Broadband networks and a new view of high performance, large scale storage-based applications. In *Proceedings of Internetworking'96*, Nara, Japan, Sept. 1996.

[17] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, pages 190–201. ACM Press, Nov. 2000. ACM SIGPLAN Notices, Vol. 35, No. 11.

[18] T. Kurc, U. Catalyurek, X. Zhang, J. Saltz, R. Martino, M. Wheeler, M. Peszyńska, A. Sussman, C. Hansen, M. Sen, R. Seifoullaev, P. Stoffa, C. Torres-Verdin, and M. Parashar. A simulation and data analysis system for large scale,data-driven oil reservoir simulation studies. *Concurrency and Computation: Practice and Experience, accepted for publication.*, 2004.

[19] S. Langella. Intellegent data management system. Master's thesis, Computer Science Department, Rensselear Polytechnic Institute, 2002.

[20] V. Matossian and M. Parashar. Autonomic optimization of an oil reservoir using decentralized services. In *Proceedings of the 1st International Workshop on Heterogeneous and Adaptive Computing– Challenges for Large Applications in Distributed Environments (CLADE 2003)*, pages 2–9. Computer Society Press, June 2003.

[21] MySql Database. *http://www.mysql.com/*.

[22] J. S. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski. The internet backplane protocol: Storage in the network. In *99: Network Storage Symposium.*, Oct. 1999.

[23] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[24] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H. Takagi. Ninf: a network based information library for a global world-wide computing infrastructure. In *Proceedings of HPCN'97 (LNCS-1225)*, pages 491–502, 1997.

[25] X. Shen and A. Choudhary. DPFS: A distributed parallel file system. In *IEEE 30th International Conference on Parallel Processing (ICPP)*, Sept. 2001.

[26] SRB: The Storage Resource Broker. *http://www.npaci.edu/DICE/SRB/index.html*.

[27] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

[28] D. Thain, J. Bent, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Pipeline and batch sharing in grid workloads. In *Proceedings of High-Performance Distributed Computing (HPDC-12)*, pages 152–161, Seattle, Washington, June 2003.

[29] B. Tierney, W. Johnston, J. Lee, G. Hoo, and M. Thompson. An overview of the distributed parallel storage server (DPSS). Available at http:// www-didc.lbl.gov/ DPSS/ Overview/ DPSS.handout.fm.html.

[30] R. Wolski, J. Brevik, C. Krintz, G. Obertelli, N. Spring, and A. Su. Running Everyware on the computational grid. In *Proceedings of the 1999 ACM/IEEE SC Conference (SC 1999)*, Portland, OR, Nov. 1999.

[31] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6):757–768, 1999.

[32] Xindice. *http://xml.apache.org/xindice/*.